Development Plan
RoCam

Team #3, SpaceY
Zifan Si
Jianqing Liu
Mike Chen
Xiaotian Lou

Table 1: Revision History

Date	**Developer(s)**	**Change**
Sept. 10, 2025	Mike Chen	Proof-of-Concept Plan and Expected Technology
Sept. 13, 2025	Jianqing Liu	Team Member Roles and Communication Plan
Sept. 13, 2025	Jianqing Liu	Expected Technology and Coding Standard
Sept. 15, 2025	Xiaotian Lou	Coding Standard for Python
Sept. 20, 2025	Jianqing Liu	Team Meeting Plan, Workflow Plan and Scheduling
Sept. 21, 2025	Xiaotian Lou	Team Charter
Sept. 21, 2025	Jianqing Liu	Misc changes and expedited workflow plan
Sept. 22, 2025	Xiaotian Lou	change on Team Charter
Sept. 22, 2025	Zifan Si	Fix development plan based on lecture slides
Sept. 22, 2025	Xiaotian Lou	update reflection

This document outlines the development plan for RoCam, a high performance vision-guided rocket tracker.

1 Confidential Information?

No confidential information to protect.

2 IP to Protect

No IP to protect.

3 Copyright License

This project adopts the MIT license, which is available at this link.

4 Team Meeting Plan

The team will meet from 7:00–8:00 PM every Tuesday and Saturday. Meetings will be held online using Discord. If an in-person session is required for hardware testing, a separate time and location will be scheduled in advance of each event.

Each team meeting will be structured as follows:

1. The agenda for each meeting will be posted as a GitHub issue ahead of time.
2. Each team member will share task updates (progress, difficulties).
3. Discuss and distribute new tasks to be worked on.
4. (Saturdays only) Draft an email for the supervisor summarizing weekly progress and any questions.

Communication with the supervisor will include a weekly update email and optional online or in-person meetings if either party requests them.

Justification: Meeting twice a week gives us enough touchpoints to stay aligned without eating into development time. By keeping careful notes and sharing them after each meeting, everyone stays up to date—even if someone can't attend—and we avoid the need for extra meetings that would slow progress.

5 Team Communication Plan

- **Discord**: General team communication, informal discussions, quick updates, and meetings.
- **GitHub**: Code-related discussions, project management, and meeting notes.
- **Zoom**: Meetings with supervisor.
- **Email**: Weekly progress updates to supervisor.

Justification: These four channels cover everything we need without overlap—fast chat (Discord), structured task tracking (GitHub), live supervisor discussions (Zoom), and formal updates (Email). Using more tools would only add confusion and split conversations across too many places, while using fewer would leave gaps in how we share information.

6 Team Member Roles

- **Project Manager**: Oversees project timeline, coordinates tasks between team members, manages deliverables and deadlines, and serves as primary point of contact with supervisor and stakeholders.
- **Meeting Chair**: Leads team meetings, prepares agendas, ensures discussions stay on track, facilitates decision-making, and manages meeting time effectively.
- **Notetaker**: Records meeting minutes, tracks action items and decisions, maintains documentation of team discussions, and distributes meeting summaries to all members.
- **Quality Assurance**: Reviews code, documentation, and deliverables for quality and consistency, conducts testing and validation, ensures adherence to coding standards and project requirements, and manages the review process for all team outputs.

Justification: These four roles cover everything our team needs: someone to coordinate and manage progress (Project Manager), someone to run organized meetings (Meeting Chair), someone to keep accurate records (Notetaker), and someone to check the quality of our work (Quality Assurance). Together, they cover planning, communication, documentation, and review, so nothing important is left out.

7 Workflow Plan

7.1 Normal Features

1. Planning
   (a) Create an issue in GitHub Projects under "Backlog" using an appropriate template (bug or enhancement), and assign it to the correct subproject.
   (b) Backlog issues will be discussed during meetings to refine scope and requirements.
   (c) If the issue is approved for development, assign an owner and a deadline, then move it into "Todo".
2. Developing
   (a) The assignee will work on the task in a new branch: [main-author-name]/[feature-name].
   (b) Move the issue into "In Progress".
   (c) (optional) If the author wants to get feedback on the code before all the changes are complete, they can create a draft pull request and request for review.
   (d) Create a pull request once the code is ready for review. The pull request should reference the original issue.
   (e) Request a review from at least one team member and ping them on Discord.
3. Reviewing
   (a) Move the issue into "In Review".
   (b) The reviewer may comment or commit directly to the feature branch.
   (c) The reviewer approves and merges the pull request.
   (d) Delete the feature branch after the pull request is merged.
   (e) Move the issue into "Done".

7.2 Minor Changes (Expedited Workflow)

For small, low-risk tasks (e.g., fixing typos or minor UI adjustments), the normal workflow may be skipped to reduce overhead. Creating an issue on the GitHub Projects board is not required. Instead:

1. Open a pull request against the main branch with "minor" label attached.
2. Obtain at least one reviewer approval before merging.
3. Delete the feature branch after the pull request is merged.

Justification: This workflow is enough to keep our work organized and traceable without being too heavy. Every task starts as an issue so nothing is forgotten, branches keep code changes separate, and pull requests with reviews make sure mistakes get caught early. Using GitHub Projects ties everything together, so planning, coding, and reviewing all stay in one place. More complicated processes would just slow us down, while skipping steps would risk bugs and confusion.

8 Project Decomposition and Scheduling

This project is decomposed into the following subprojects:

- **Web App**: responsible for remote management.
- **CV Pipeline**: responsible for locating the rocket.
- **Motion Control**: responsible for controlling gimbal movement.

All code for the subprojects, along with documentation, is centralized in a single monorepo here. GitHub Projects is used for project management and can be accessed here.

While development will be broken down into smaller features with individual deadlines, the overall project will follow the major deadlines below.

Table 2: Major Deliverables

Date	Deliverable	Files
Sept. 22, 2025	Problem Statement, POC Plan, Development Plan	Problem Statement POC and Development Plan
Oct. 6, 2025	Req. Doc. and Hazard Analysis Revision 0	Req. Doc. Hazard Analysis
Oct. 27, 2025	V&V Plan Revision 0	V&V Plan
Nov. 10, 2025	Design Document Revision -1	Design Document
Nov. 17-28, 2025	Proof of Concept Demonstration	
Jan. 19, 2026	Design Document Revision 0	Design Document
Feb. 2-13, 2026	Revision 0 Demonstration	
Mar. 9, 2026	V&V Report and Extras Revision 0	V&V Report Extras
Mar. 23-29, 2026	Final Demonstration (Revision 1)	
TBD	EXPO Demonstration	
Apr. 6, 2026	Final Documentation (Revision 1)	

Justification: Splitting the work into Web App, CV Pipeline, and Motion Control is enough because these three parts match the main technical challenges of the project. Each subproject has clear boundaries, so team members can work in parallel without confusion. Putting everything in a single monorepo keeps the codebase consistent, and using GitHub Projects ties scheduling directly to the actual code, so we don't need extra tools that could complicate tracking.

9 Proof-of-Concept Demonstration Plan

The following are the planned steps of the POC:

1. Acquire an initial image from a camera with a stationary target.
2. Activate the system's tracking mode. It will segment and detect multiple moving objects in the image.
    - Segmentation and moving-object detection will use computer vision techniques.
    - The computer vision model will be deployed on a Jetson Nano.
3. The user selects a stationary or moving object as the target.
4. As the target moves, the system keeps it centered in the image for smooth tracking.
    - The system will handle occlusion and temporary loss of the target.
    - The user can manually reselect the target if needed.
    - Real-time camera control will be implemented using an STM32 microcontroller.

The following is a list of primary risks to consider for the POC:

1. The computer vision system may not process images at a sufficient frame rate.
    - If this occurs, we will optimize the existing model, consider using a more powerful board, lower the frame rate, or use a traditional algorithmic approach that detects motion via pixel-wise image comparison.
    - We also reserve the option to use models trained for a specific set of objects to increase throughput.
2. The STM32 may not deliver real-time control to the camera.
    - If this occurs, we will consider using a more powerful microcontroller, or a different control technique.
3. Integration of the frontend, backend, computer vision model, and microcontroller may be more difficult than expected.
    - If this occurs, we will consider using a more powerful integrated single-board computer, deploying via cloud computation, or redesigning our control flow to simplify integration.

Other smaller risks to consider:

1. UI usability issues: The user interface may not be intuitive or easy to use, leading to user frustration or errors.
    - Potential solution: Conduct user testing and gather feedback to improve the interface design.

Justification: These steps give us just enough to prove the idea works without overloading the demo. Taking a picture, tracking movement, picking a target, and keeping it in view are the core things we need to show. We also listed the biggest risks so we're ready if something goes wrong. This way the POC is doable, but it still shows the hardest and most important parts of the system.

10 Expected Technology

- Motion Control
  - STM32 Microcontroller
  - Language: Rust
  - Framework: embassy-rs
  - Formatting: rustfmt
  - Linter: rust-clippy
  - Unit Testing: Rust built-in
  - Code Coverage: grcov
- Computer Vision
  - NVIDIA Jetson
  - NVIDIA JetPack
  - Language: Python
  - Libraries: OpenCV, NumPy, Matplotlib, Torch
  - Open Source Models: Ultralytics YOLO, SAM, various ViTs
  - Formatting: Black
  - Linter: pylint and ruff
  - Unit Testing: pytest
  - Code Coverage: coverage
  - Containerized using Docker
- Web App
  - Web Server: Flask
  - Language: TypeScript
  - Framework: React
  - Formatting: prettier
  - Linter: eslint
  - End-to-end Testing: Cypress
- All of the above will use GitHub Actions for CI.
- Development Tools
  - VS Code
  - PyCharm
  - Git
  - GitHub

11 Coding Standard

- Rust: The Power of 10 Rules
- Python: PEP 8
- TypeScript: typescript-eslint

Justification: These standards are enough to keep our code clean and consistent without adding extra rules that slow us down. The Power of 10 helps make Rust code safer, PEP 8 is the common style guide for Python, and typescript-eslint is widely used for TypeScript. Sticking to these well-known guides means anyone reading or reviewing our code will find it easy to follow.

Appendix — Reflection

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. Why is it important to create a development plan prior to starting the project?
2. In your opinion, what are the advantages and disadvantages of using CI/CD?
3. What disagreements did your group have in this deliverable, if any, and how did you resolve them?

Development Plan Reflection — Jianqing Liu

1. *Why is it important to create a development plan prior to starting the project?*

   I'm responsible for drafting most of the sections of this deliverable except Proof-of-Concept Demonstration Plan and part of Expected Technology. One section I want to highlight is "7.2 Minor Changes (Expedited Workflow)". I got this idea from my time working in a coop position, where the high overhead of project management prevented me from fixing small issues I encountered during development, as the fix was unrelated to the feature I was working on. I hope this will not turn into a loophole where all the pull requests are labeled as "minor" to bypass the normal planning process.

   I also didn't realize the usefulness of the development plan until I started working on it. It makes a lot of implicit assumptions explicit (especially the workflow plan and team meeting plan, which were only communicated verbally in my previous positions), and helps the team better collaborate.

2. *In your opinion, what are the advantages and disadvantages of using CI/CD?*

   The advantages of CI far outweigh the disadvantages. The most important use for it in my opinion is to enforce coding standards and catch regressions early. The only disadvantage I can think of is the extra overhead of setting up CI, but it is worth it.

   CD is a nice to have, but sometimes not feasible. For our project, since we need to deploy to actual hardware (Jetson and STM32), it is not feasible to set up CD. However, we can create a deployment script to simplify the process.

3. *What disagreements did your group have in this deliverable, if any, and how did you resolve them?*

   We did not have any disagreements in this deliverable.

Development Plan Reflection — Xiaotian Lou

1. *Why is it important to create a development plan prior to starting the project?*

   A development plan exposes overlooked details early and clarifies responsibilities. By mapping objectives to work packages, timelines, resources, and risks, it aligns expectations across the team, reduces ambiguity in day-to-day decisions, and provides a baseline for tracking progress and adjusting course when constraints change.

2. *What disagreements did your group have in this deliverable, if any, and how did you resolve them?*

   We did not encounter material disagreements at this stage. If anything, the team is slightly over-optimistic—great for momentum but it can hide risk. To balance this, we added explicit buffers, milestone checkpoints, and a short risk log so enthusiasm is paired with realistic schedules and visible contingency plans.

3. *What disagreements did your group have in this deliverable, if any, and how did you resolve them?*

   We did not have any disagreements in this deliverable.

Development Plan Reflection — Shike Chen

1. *Why is it important to create a development plan prior to starting the project?*

   I have helped in drafting and revising the development plan. I realized the importance of the development plan as I was working on the proof of concept. In the process of writing the proof of concept, I went through many risks of the project that I had not thought of before. I believe the development plan is vital at the beginning of the project, as it helps the team identify the risks and feasibility of the project.

2. *In your opinion, what are the advantages and disadvantages of using CI/CD?*

   The largest advantage of CI/CD is that each small update, improvement, feature, or bug fix can be integrated into the development environment as soon as it is ready. This greatly enhances the flexibility and robustness of the product. The main disadvantage of CI/CD is that it requires a lot of setup and, in some cases, a mistake merged into the main branch can lead to large problems.

3. *What disagreements did your group have in this deliverable, if any, and how did you resolve them?*

   We did not have any disagreements in this deliverable.

Appendix — Team Charter

External Goals

Our team's primary goal is to learn something practical and transferable skills on full software development lifecycle that can be applied in the workforce, including requirements engineering, UI design, API design, iterative prototyping, and validation.

Additionally, we aim to create a project that we can confidently discuss in interviews, demonstrating our ability to work on real-world problems. While we focus on personal and professional growth, we also aim for an A+ as a nice-to-have achievement.

Attendance

Expectations

Members commit to scheduled meetings, arrive on time, and participate fully. If a member cannot attend, they will notify the team in advance, provide clear reasons, and help coordinate a new time if full attendance is required. Repeat lateness or unexplained absence will be addressed promptly to protect cadence.

Acceptable Excuse

Acceptable reasons include emergencies, illness, family obligations, or other significant duties, provided the team is informed as early as possible. Vague or last minute reasons (e.g., "forgot") or avoidable conflicts are not accepted, as they jeopardize schedule, trust, and safety readiness.

In Case of Emergency

When emergencies prevent attendance or delivery, the member must notify the team immediately via our primary channel (Discord). The team may redistribute tasks or reschedule as needed. If a deadline is impacted, notify the team and the instructor or TA promptly so proper arrangements are made without risking progress or stakeholder expectations.

Accountability and Teamwork

Quality

Our team sets explicit expectations for meeting prep and deliverable quality, with safety and reliability emphasized for field operations:

- **Meeting Preparation**:
  - Review materials and complete assigned work before meetings; bring concise, verifiable updates with blockers and proposed options.
  - Keep discussions decision oriented; capture risks, assumptions, and next actions; timebox and create follow ups for deep dives.
  - Maintain a decision log and lightweight RACI notes for key items.
- **Deliverables Quality**:
  - Meet standards of correctness, clarity, completeness, and reproducibility; include error and empty states in the UI.
  - Align with API contracts and data schemas; document versions and backward compatibility expectations.
  - Provide evidence: latency/FPS snapshots, basic load trials, and stability checks (e.g., 30+ min runs) with notes on limits.
- **Verification and Safety Readiness**:

  - Validate with staged tests: synthetic videos, recorded flights, and hardware in the loop where feasible.
  - Track KPIs: end to end latency (p50/p95), FPS, target reacquisition time, crash free session rate, and UI task success.
  - Record assumptions, known issues, and mitigations for field usage.
- **Accountability and Feedback**:
  - Own outcomes; escalate early if help or time is needed; provide timely handoffs and keep docs current.
  - Welcome review feedback; address comments within 7 days unless a different SLA is agreed in advance.

By upholding these practices, meetings remain efficient and deliverables remain professional, testable, and field minded.

Attitude

We adopt the following **expectations** for contribution, interaction, and cooperation to sustain a respectful and high performance culture:

- **Respectful Communication**: Listen first; respond with concrete specifics; separate people from problems.
- **Open Collaboration**: Share ideas and risks early; offer support during integration and time critical milestones.
- **Accountability**: Meet timelines or replan early with options that protect scope, safety, and critical path.
- **Positive Attitude**: Stay solution oriented under constraints; help teammates recover when issues occur.
- **Commitment to Quality**: Prefer simple, observable solutions and incremental improvement over risky big bangs.

We further adopt a concise **code of conduct**:

- **Inclusivity**: Welcome diverse backgrounds and viewpoints; ensure space for every member to contribute.
- **Professionalism**: Act with integrity; avoid offensive language and behavior in all settings.
- **Collaboration and Feedback**: Keep critique about the work, not the person; make feedback specific and actionable.
- **No Tolerance for Harassment**: Any harassment is unacceptable and will be reported and handled immediately.

Conflicts are handled via the following **resolution plan**:

1. **Address the Issue Directly**: Involved parties first attempt a respectful, solution oriented discussion.
2. **Mediation by a Neutral Member**: If unresolved, a neutral teammate facilitates and seeks common ground.
3. **Escalation to Instructor/TA**: If still unresolved, we escalate for guidance and a final decision.
4. **Follow Up and Monitoring**: Verify the resolution holds; refine norms if needed to prevent recurrence.

By following these expectations and processes, we maintain a positive and collaborative environment while delivering course outcomes.

Stay on Track

To keep momentum and visibility while preparing for safe field demos, we will use the methods below:

1. **Regular Check ins and Progress Updates**: Hold *weekly meetings* where each member shares progress, risks, and next steps; capture actions, owners, and due dates.
2. **Performance and Reliability Metrics**:
   - *Attendance* tracked in GitHub Issues or the project board.
   - *Commits/PRs* for steady, reviewable progress with CI gates.
   - *Task on time rate* and defect turnaround time per milestone.
   - *Runtime KPIs*: latency, FPS, time to reacquire, stability.
3. **Rewards for High Performers**: Recognize members who meet or exceed expectations with public kudos and opportunities to lead desired submodules or integrations.
4. **Managing Underperformance**: If a member underdelivers for more than 3 weeks:
   - Start with a team conversation to understand obstacles, add pairing, and set concrete checkpoints.
   - If issues persist, add guardrail tasks; in severe cases, meet with the TA or instructor.
5. **Consequences for Not Contributing**: When a member does not contribute fairly:
   - Rebalance workload and expectations to protect the milestone.
   - Escalate to the TA or instructor if patterns continue.
6. **Incentives for Meeting Targets Early**: Members who reliably meet or exceed targets receive first pick for next milestone tasks and additional leadership opportunities.

Team Building

We host a light, bi weekly hangout to build trust and informal bandwidth. Examples include attending campus events, shared meals, or quick bubble tea breaks at inclusive times. Optional pair programming or design jam sessions may be used to accelerate learning and cohesion.

Decision Making

We prefer consensus based decisions and ensure everyone has a chance to speak before concluding. If consensus is not reachable in time, we vote with equal weight per member and accept the majority decision. Summaries and action items are posted within 24 hours in Discord and linked Issues.

To handle disagreements: address issues directly and respectfully.

1. Allow members to share concerns without interruption and record key points neutrally for transparency.
2. Keep discussion focused on the topic, not on personal attributions or intent.
3. When needed, appoint a neutral facilitator to guide the conversation toward a resolution within the timebox.
4. If the issue persists, revisit project goals, safety constraints, and stakeholder needs; choose the option that best aligns with them.

By applying these strategies, we preserve a collaborative environment and make timely, transparent decisions aligned with a high quality tracking solution.